MOLECULAR DYNAMICS SIMULATION OF SEDIMENTATION USING THE MONOTONIC LAGRANGIAN GRID

(4)

AR-005-727

R.A.J. BORG, D.A. JONES AND S.G. LAMBRAKOS

MATERIALS RESEARCH LABORATORY
DSTO

MOLECULAR DYNAMICS SIMULATION OF SEDIMENTATION USING THE MONOTONIC LAGRANGIAN GRID

R.A.J. Borg, D.A. Jones and S.G. Lambrakos*

MRL Technical Report
MRL-TR-89-30

ABSTRACT

We describe the implementation of a new algorithm for tracking nearest-neighbours, the Monotonic Lagrangian Grid (MLG), in a recently developed Molecular Dynamics (MD) code used to study sedimentation phenomena. The MLG is a highly efficient algorithm for tracking particle positions which is based on the assignment of a set of indices to each particle in the system. These indices are ordered according to the relative positions of each of the particles, and are continuously updated as particles move around in space. Application of the MLG to the existing sedimentation code reduces the dependence of the CPU time on total particle number N from N^2 to N. This is the first application of the MLG to this type of system. Areas of the code are identified in which implementation of more advanced aspects of the MLG algorithm would allow efficient vectorization of the code and result in additional reduction in CPU time. Other applications for MD codes at MRL based on the MLG are also described.

* Laboratory for Computational Physics and Fluid Dynamics
Naval Research Laboratory, Washington DC 20375, USA

Published by DSTO Materials Research Laboratory
Cordite Avenue, Maribyrnong, Victoria 3032, Australia
Telephone: (03) 319 3887
Fax: (03) 318 4536

AR No. 005-727

Approved for public release

CONTENTS

MOLECULAR DYNAMICS SIMULATION OF SEDIMENTATION USING THE MONOTONIC LAGRANGIAN GRID

1. INTRODUCTION

A Molecular Dynamics (MD) code was recently written by one of us to study the sedimentation behaviour of solid RDX particles in liquid TNT [1]. This process occurs during the cast filling of military ordnance with the explosive Composition B, a mixture of 60% RDX and 40% TNT. One of the major applications planned for the code was to study the effect of the RDX particle size distribution on the sediment density. Simulations were reported for suspensions with a maximum number of 200 particles. When all particles were the same size the calculations showed that the packing density was always less than the equivalent close packed structure and depended on the particle radius, with an increase in radius leading to a decrease in sediment density. When the sample consisted of particles of two distinct sizes then fractionation was found to occur. In all cases, the larger particles settled before the smaller particles, giving rise to a layer of small particles on top of the sediment.

Simulations of realistic particle size distributions with the code reported in [1] are not possible because of the restriction on the total number of particles which can be conveniently calculated. This is due to the relatively inefficient algorithm used for tracking particles and computing interparticle forces. In the method described in [1] the total computational time increases as N^2, where N is the total number of particles in the simulation.

Various algorithms have been devised to reduce this N^2 dependency. Scalar sorting procedures have been developed which scale linearly with N or $N\ell nN$ [2], and vectorizable algorithms using neighbour list techniques have also been developed [3]. None of these methods is particularly suitable however for large three-dimensional simulations or vector supercomputers. The cost of the scalar methods is still prohibitive for large simulations, and the neighbour list techniques have large storage requirements and are also not ideal for large systems. None of these techniques has been applied to the code described in [1].

A new algorithm for keeping track of near neighbour relationships has recently been described [4, 5]. The scheme is based on constructing a "monotonic Lagrangian grid" (MLG), a data structure where adjacent particles in space have close grid indices. The computational cost of the scheme scales as N, and the algorithm

vectorizes easily using data from contiguous memory locations. The MLG has already been applied to several MD calculations and has been shown to be very efficient when large numbers of particles are used [6, 7, 8].

The purpose of this communication is to report on our progress in implementing the MLG algorithm into the code reported in [1]. We are interested in the MLG in particular for several reasons:

(i) Since writing the code described in [1] we have gained access to a CRAY X-MP/14 supercomputer. The architecture of this machine is significantly different from that of the VAX 8700 computer on which the code was first developed. One of the limitations on the computational speed of a VAX, or any scalar computer, is that a separate instruction is required each time a number is moved from the memory to the Central Processor Unit (CPU). The CRAY overcomes this limitation by utilizing a pipeline architecture. Computations on elements within the pipeline are performed more quickly than in scalar mode because successive elements in the pipeline are automatically moved into the CPU without the need for a separate instruction for each element. Increases of fifty or more in overall computational speed are easily obtained for programs which are designed to make optimum use of this pipeline architecture. Obviously any changes we make to the code should be capable of fully utilizing this feature (such a code is said to be fully "vectorized"). The MLG algorithm is ideally suited to this task.

(ii) We have in mind other applications for MD codes at MRL which would benefit considerably from the use of the MLG. We have for some time been interested in the general area of high velocity impact and penetration. To date our approach to this work has been through the use of large Eulerian or Lagrangian finite difference codes such as the HULL code developed by Systems, Science and Software for the US Air Force, and now maintained by Orlando Technology. Such codes are not the optimum approach however when large amounts of fragmentation or cratering occur. In these cases a particle approach such as the Smooth Particle Hydrodynamics (SPH) scheme described by Monaghan seems more appropriate [9]. In the SPH method the continuum equations of hydrodynamics are replaced by particles which simulate the fluid flow. Again, the problem of tracking particles, and their nearest neighbours, appears. SPH codes have been run on a CRAY supercomputer using a linked list algorithm for tracking nearest neighbours, but large amounts of storage were required [10]. It seems likely that the efficiency of SPH codes could be improved significantly by the addition of MLG algorithms. We are currently evaluating the possibility of work along these lines.

(iii) Another area of more immediate interest is the simulation of shock waves in solids. We are currently developing an experimental facility to make precise measurements of shock Hugoniots in selected materials [11]. Concurrent with this experimental work we intend to calculate some Hugoniots from microscopic theories using ab-initio potentials. This will require a knowledge of the shockfront thickness, and we intend to calculate this using an MD code to simulate the propagation of the shock through the crystal. Our initial interest will be in alkali halide crystals, in which case the MLG will be of no use to us. The alkali halides, being ionic crystals, have long range Coulomb potentials which require calculation of Ewald sums, and all particles must be considered in the force calculation. The MLG will become of great benefit when we consider extensions to molecular crystals, where only near neighbour interactions will need to be considered.

The objective of the work reported here was to demonstrate that the sedimentation code could be implemented in MLG form, and to reduce the CPU time for a given number of cycles from an N^2 dependency to one which scales linearly as N. In this we have been successful. Further improvements could be made, particularly with regard to vectorization of the code.

2. THE MLG SEDIMENTATION CODE

A detailed description of MLG algorithms has already been given by Boris [4], and Lambrakos and Boris [5]. Here we briefly review those aspects of the MLG relevant to the sedimentation code. Consider N particles randomly distributed in three-dimensional space. For many physical systems, including the sedimenting system studied here, the motion of any given particle is largely unaffected by most of the N-1 other particles in the system. Typically, the particle will be significantly influenced by only a relatively small number of nearby particles. These are defined to be the particles' "nearest neighbours". Finding the nearest neighbours in an efficient manner is a central part of any MD simulation. A natural choice is to calculate the distance between the particle and the N-1 other particles in the system. A cutoff radius R_c is then defined such that particles which are separated by a distance greater than R_c have negligible effect on one another. The time taken to calculate the force on the given particle is then reduced because only those particles within the cutoff radius R_c need to be considered. The difficulty with this approach is that calculating the distance to each of the N-1 other particles is almost as expensive as including all N-1 particles in the force calculation.

The MLG algorithm was designed to be a highly efficient method for solving this nearest neighbours problem. The data structure is arranged so that a particle's nearest neighbours are readily identified, but without the need to continually calculate the distance to each of the other N-1 particles. The basic idea of the MLG is to assign to each particle a set of three indices, (i, j, k), so that particles which are near one another in real space have indices differing from one another by only small integer values. Let the x, y and z coordinates of a particle with MLG indices (i, j, k) be denoted by X(i,j,k), Y(i,j,k) and Z(i,j,k). The assignment of the MLG indices to the N randomly distributed particles is carried out according to the following monotonic prescription:

$$\begin{aligned} &X(i,j,k) \le X(i+1,j,k) \quad \text{for } 1 \le i \le NX - 1 \\ &Y(i,j,k) \le Y(i,j+1,k) \quad \text{for } 1 \le j \le NY - 1 \\ &Z(i,j,k) \le Z(i,j,k+1) \quad \text{for } 1 \le k \le NZ - 1, \end{aligned} \tag{1}$$

where N = NX x NY x NZ. This ordering guarantees that particles which are near one another in space have close grid indices. Hence a particle's nearest neighbours can always be found by considering small index offsets, rather than by imposing some condition on the separation of particles in space. If a maximum index offset N_c is defined, then computations for each particle are only made for those particles with index offsets less than N_c. In the computer code itself the nearest neighbours are identified by constructing a nearest-neighbours template (NNT) based

on this maximum index offset. The NNT specifies exactly which nearest-neighbours will be used in the force calculations, and is defined by a main grid location and a fixed number of close locations defined by N_c. The size of the NNT will depend on the particle density and the relative level of order in the system. N_c typically has a small integer value in the range 3 to 7, although the selection of the value to be used for a given calculation usually requires some experimentation.

As the system evolves in time and particles pass one another in space the monotonic mapping of positions onto grid indices must be continuously updated to maintain the correct ordering. This is done by a process known as "swapping". As two particles move past each other in real space the information describing these two objects is moved to different locations in computer memory. Swapping is an iterative process, and the number of iterations will depend on the extent to which the particle motion disrupts the monotonic indexing. For accurate integration of the equations of motion two particles will generally move far less than a typical separation distance per timestep. Under these conditions only slight disruption of the MLG indexing occurs and very few swaps are required to restore correct MLG order. In the code itself a swap is executed by checking to see if there are any violations of the conditions imposed by equation (1). If a violation is found for a particular set of indices then all the information describing the two objects is exchanged between the two grid locations. This scheme has been shown to require very few swaps per timestep, and the relative computational cost of the swapping has been found to be very small [4, 5].

The structure of a typical MLG MD code is shown in Figure 1. First an initialization subroutine is called to assign positions and velocities to each of the particles. Next the iterative swapping process is performed to place all arrays in MLG order, and then a nearest-neighbours template (NNT) is defined. The time evolution of the system can then commence. Separate subroutines first calculate the total force on each particle in the system, and then integrate the equations of motion over a time step Δt. Boundary conditions are then applied. Next a monotonicity check is made to see if the system is still in MLG order. If it is, the time step is updated and the cycle is repeated. If not, the iterative swapping process is performed until MLG order is restored, the time step is updated, and the cycle repeats.

Rather than trying to convert the sedimentation code described in [1] into MLG form, it was easier to start from a basic MLG code having the structure shown in Figure 1, and then transfer appropriate subroutines from the old code into the MLG code. The more important of these were the subroutine to calculate the force arrays, and the subroutine to integrate the equations of motion.

The code described in [1] simulates the motion of N "macroscopic" particles (i.e. having particle sizes from 10 μm to 100 μm) in a viscous medium subject to the force of gravity, and to their own mutual repulsion when in contact. The particles are confined to a container of definite dimensions, and a particle/floor force is defined so that the particles will eventually come to rest at the bottom of the container. Gravity is taken to act in the negative z direction, and periodic boundary conditions are applied in the x and y directions.

A Lennard-Jones (LJ) potential was used to calculate the interparticle forces $\mathbf{F}_{ij}$.

$$\mathbf{F}_{ij} = \left[\left(\frac{A_i A_j}{r_{ij}}\right)^{12} - \left(\frac{C_i C_j}{r_{ij}}\right)^{6}\right] \frac{\hat{\mathbf{d}}}{r_{ij}} , \qquad (2)$$

where $\hat{\mathbf{d}}$ is a unit vector along the interparticle line, r_{ij} is the particle distance between particles i and j, and the A_i and C_i are constants. This particular form of the force allowed for the possibility of a mixture of particles of different sizes, and the splitting of the force constant terms into the product of two one-dimensional arrays aided vectorization of the code. The constants A_i and C_i are different from the LJ parameters used in [1], and their values are given in Table 1. The use of a Lennard-Jones potential is more common in atomic systems, but it was also found to be very suited to the sedimenting system studied here and in [1]. The particles are assumed to be "soft" spheres. This was necessary to overcome numerical stability problems, as well as reducing particle overlap. In [1] the force was calculated using a table look-up, and was set to zero unless the particles were touching or overlapping. In the code described here the force was calculated from equation (2) for each particle for each time step, and no cutoff distance was used, the range of the force being governed by the size of the NNT, which is discussed in the next paragraph.

The viscous force acting on each particle is the same as in [1] and is given by

$$\mathbf{F}_{iv} = - 6\pi\eta r_i \mathbf{v}_i , \qquad (3)$$

where η is the liquid phase viscosity, r_i is the radius of particle i, and $\mathbf{v}_i$ is the velocity of particle i.

The particle-floor force has the form

$$\mathbf{F}_i^{floor} = \left(\frac{K_i}{r_i^{floor}}\right)^{13} \hat{\mathbf{k}}, \qquad (4)$$

where r_i^{floor} is the distance between particle i and the bottom of the container, K_i is a constant for particle type i, and $\mathbf{k}$ is a unit vector in the z direction. If $r_i^{floor} > r_i$ we set $\mathbf{F}_i^{floor} = 0$. The K_i values are given in Table 1.

Table 1 Values of constants used in the MLG code

Particle radius (μm)	A_i ($J^{1/24}\ m^{1/2}$)	C_i ($J^{1/12}\ m^{1/2}$)	K_i ($N^{1/13}$ m)
30	2.20×10^{-3}	2.54×10^{-4}	6.48×10^{-6}
50	3.20×10^{-3}	5.28×10^{-4}	1.21×10^{-5}

We found little difficulty in getting the sedimentation code running in MLG form, although some care had to be exercised in the choice of the NNT. The method of choosing an NNT is to first choose a template which is obviously too big. Once the code is working satisfactorily with this NNT then gradual reductions in the size of the template can be made until problems are encountered. At this stage the NNT is too small and needs to be enlarged slightly. Further runs are then made with the NNT at this size to check that it is optimum for the system being studied. We used an NNT almost identical to the one used in previous applications of the MLG to atomic and molecular systems [6, 8], with a maximum index offset value of $N_C = 3$. This was found to give acceptable results for all the simulations performed, and for the range of densities encountered.

3. RESULTS

Having found a suitable size for the NNT and established that the sedimentation code worked in MLG form, our next objective was to investigate the timing characteristics of the code. To do this we made a series of runs with both the old and the new codes for increasing numbers of particles. The results are shown in Table 2 and Figure 2. The total CPU time for each run has been split into the time taken for the first cycle, and then the time taken for a further 100 cycles. We did this because we found that the time taken for the initial placement of the particles in the container was taking an appreciable fraction of the CPU time for such short runs, and was preventing us from making reliable timing estimates. Initially, the particles are placed randomly in the box such that no two particles overlap. As the number of particles increases, many more tries are needed to find empty spaces which the particles can occupy, and the time taken to place them randomly can increase dramatically. This is shown quite clearly in the data presented in Table 2.

Table 2 Timing Comparison Runs

Number of Particles	CPU Time (seconds)					
	Old Code (VAX)		MLG Code (VAX)		MLG Code (CRAY)	
	1 cycle	100 cycles	1 cycle	100 cycles	1 cycle	100 cycles
125	24	47	6.5	143	0.94	18
216	31	135	46	272	7.5	31
343	64	335	83	431	11	49
512	225	698	289	647	41	73
729	372	1761	332	956	48	102
1000	981	2805	671	1341	69	138

Plots of run time for 100 cycles versus total number of particles are shown in Figure 2 for both the old code and the MLG code on a VAX 8700, and for the MLG code on a CRAY X-MP/14. The old code shows the N^2 dependency (gradient equals 1.98), as previously observed [1], while the MLG code on the VAX shows a linear dependence on N (gradient equals 1.08), as expected. It is interesting to note that the MLG code on the VAX takes longer to run 100 cycles than does the old code, when small particle numbers are considered. This is not unexpected because of the additional initial computational overheads associated with the MLG, and the use of equation (2) instead of a look-up table. Crossover occurs around N = 500, and for N = 1000 the MLG version is more than twice as fast. It should be noted that the MLG was devised with particle numbers of order 10,000 or greater in mind.

The MLG code when run on the CRAY also shows a linear dependence on N (gradient equals 0.97). Independent benchmark runs on the VAX 8700 and CRAY X-MP/14 of a code which is only slightly vectorizable show an increase in speed on the CRAY of a factor of 8 [12]. As N increases from 125 to 1000 the ratio of VAX time to CRAY time increases slightly from 8 to almost 10, indicating that the vectorization capabilities of the MLG are only just beginning to be utilised. This is certainly not surprising considering the relatively small number of particles we are using. In an MLG code based on a "regular MLG" structure all of the computationally intensive parts of the code are contained within a nested set of DO loops over the indices i, j and k. In our sample of 1000 particles we have NX = NY = NZ = 10, which means that the innermost loop has a count of only 10. On a CRAY computer however it is only the innermost loop which vectorizes, and a count of 10 is far too short for the increase in speed due to the pipeline architecture to be effective. A simple way of overcoming this problem is to use a "skew-periodic" MLG structure [4,5]. In essence, this maps a plane of points onto a single vector, so that the innermost loop now has a count of, say, NX x NY. For N = 512 this would result in vectors of length 64, which is a near optimum length for a CRAY computer. This was not done in the present study however.

It should be noted that there are physical situations in which very efficient vectorization of MLG codes could be achieved without the need to implement a skew-periodic MLG. One of these is the application to a shock wave in a crystal mentioned in the Introduction. In this situation there will generally be many more atoms in the direction of propagation of the shock than at right angles to it. By choosing the innermost MLG index to index particles in this direction the innermost loop will be the largest loop and efficient vectorization will result.

We also made several production runs of the new code on both the VAX and the CRAY. It was important to do this to check that the NNT was working correctly as the density increased in the sediment phase, and also to fine tune the constants A_i and C_i. We used a total number of 512 particles, 307 with a radius of 30 μm, and 205 with a radius of 50 μm. Complete sedimentation occurred in all cases within 6,000 cycles, using a fixed time step of 200 μs. Total run time agreed closely with estimates from the 100 cycle runs.

Figure 3 shows the probability of finding a particle as a function of particle height above the floor of the container for one of the VAX simulations. This illustrates the amount of fractionation which has taken place. We are much more likely to find a larger particle at the bottom of the container than a smaller one, and we have zero probability of finding a large particle at a height greater than 320 μm. Consequently, a layer of 30 μm particles will be formed on top of the sediment. These results are similar to the ones reported in [1] for the 200 particle runs.

4. CONCLUSION

We have rewritten the MD code described in [1] to include the MLG algorithm described in [4] and [5]. As a result, the CPU time for a given number of cycles has been reduced from an N^2 dependency to one which scales linearly with N. This reduction in computing time has enabled us to make a number of production runs with up to 512 particles, whereas with the previous code we were effectively restricted to a maximum of 200. Even with this improvement, the code would still be time consuming and expensive to run if realistic particle size distributions were employed. A further increase in the speed of the code could be made if the regular MLG were replaced with a skew-periodic MLG. This would allow very efficient vectorization of the code and would increase speed by up to an order of magnitude. Improvements in speed could also be made if a table look-up were used instead of equation 2, although this would not be as simple to implement as in the earlier version of the code [1], as the MLG data structure requires a gather operation to be performed if a table look-up is used. If each of these programming techniques could be implemented in the code, then simulations of realistic particle size distributions could be made.

5. ACKNOWLEDGEMENTS

We thank Dr D.D. Richardson of MRL for his interest in this area and for providing the opportunity to undertake this work. DAJ would like to thank Dr J.P. Boris, Head of the Laboratory for Computational Physics and Fluid Dynamics at NRL, for the opportunity to learn something of MLG techniques while he was a Visiting Scientist at LCP & FD during the period February 1987 to May 1988.

6. REFERENCES

1. Borg, R.A.J. (1989). Simulation of sedimentation by molecular dynamics (MRL Report MRL-RR-7-89). Maribyrnong, Vic.: Materials Research Laboratory.

2. Hockney, R.W. and Eastwood, J.W. (1981). Computer simulation using particles, Chap. 8, pp. 267. New York: McGraw Hill.

3. Van Gunsteren, W.F. et al. (1984). Journal of Computational Chemistry, 5, 272.

4. Boris, J. (1986). A vectorized near neighbors algorithm of order N using a monotonic logical grid. Journal of Computational Physics, **66**, 1-20.

5. Lambrakos, S.G. and Boris, J.P. (1987). Geometric properties of the monotonic lagrangian grid algorithm for near neighbor calculations. Journal of Computational Physics, **73**, 183-202.

6. Lambrakos, S.G., Boris, J.P., Guirguis, R.H., Page, M. and Oran, E.S. (1989). Molecular dynamics simulation of $(N_2)_2$ formation using the monotonic lagrangian grid. Journal of Chemical Physics, **90**, 4473-4481.

7. Lambrakos, S.G., Oran, E.S., Boris, J.P. and Guirguis, R.H. (1988). Molecular dynamics simulation of shock-induced detonations in energetic solids. S.C. Schmidt and N.C. Holes (Eds.) Shock waves in condensed matter 1987, Elsevier Science Publishers B.V., pp. 499-502.

8. Lambrakos, S.G., Peyrard, M., Oran, E.S. and Boris, J.P. (1989). A new approach to molecular dynamics simulations of shock-induced detonations in solids. Physics Review B, **39**, 993-1005.

9. Monaghan, J.J. (1982). Why particle methods work. SIAM Journal of Scientific and Statistical Computing, **3**, 422-433.

10. Benz, W., Marr, D.R. and Kidman, R.B. A fast smooth particle hydrodynamics code, preprint.

11. Richardson, D.D., Northeast, E.D. and Ryan, P.F.X. (1988). An exploding foil flying plate generator (MRL Report MRL-R-1133). Maribyrnong, Vic.: Materials Research Laboratory.

12. Kummer, R.J. and Smith, D.L., private communication.

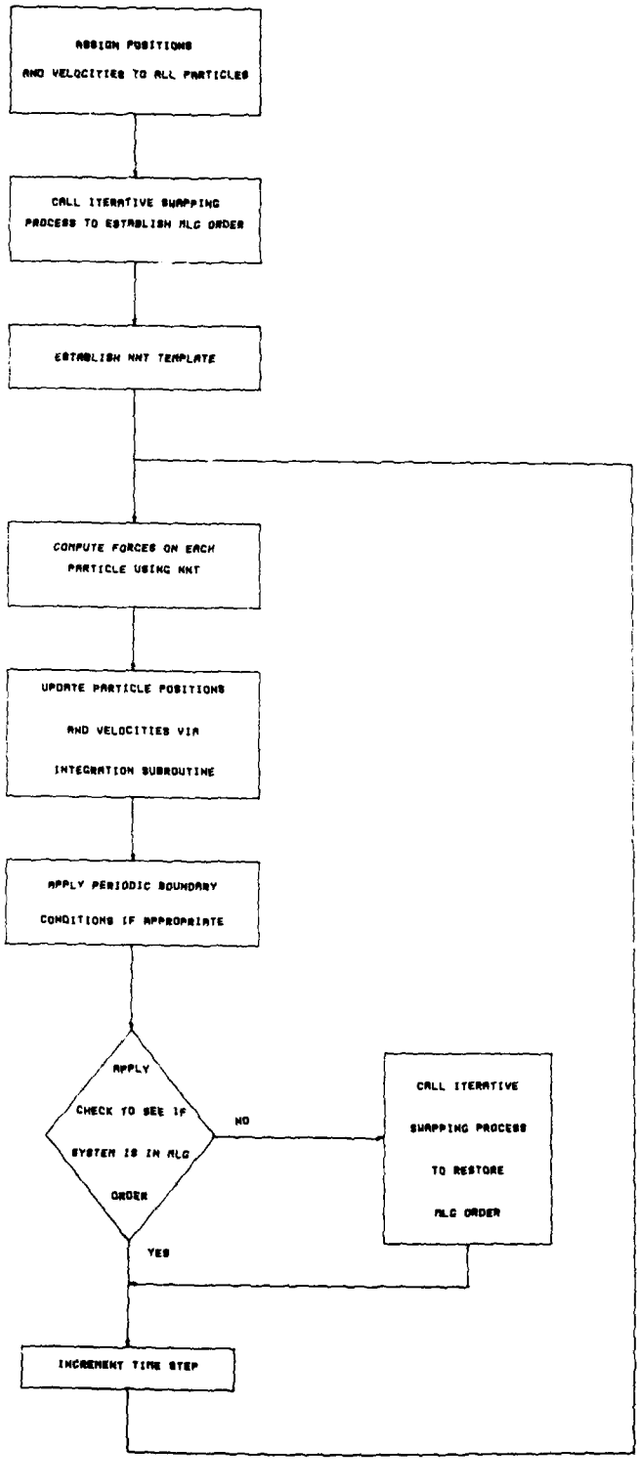

Figure 1 Flow chart for a typical MLG based MD code.

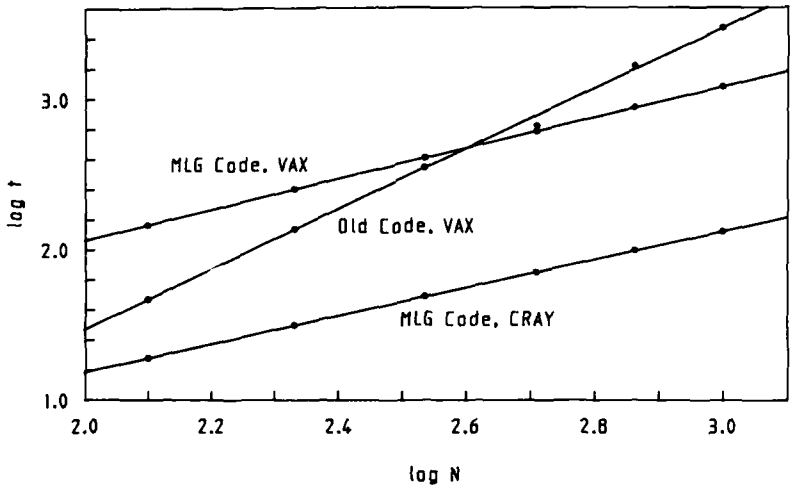

Figure 2 *CPU time for 100 cycles versus total number of particles in the simulation.*

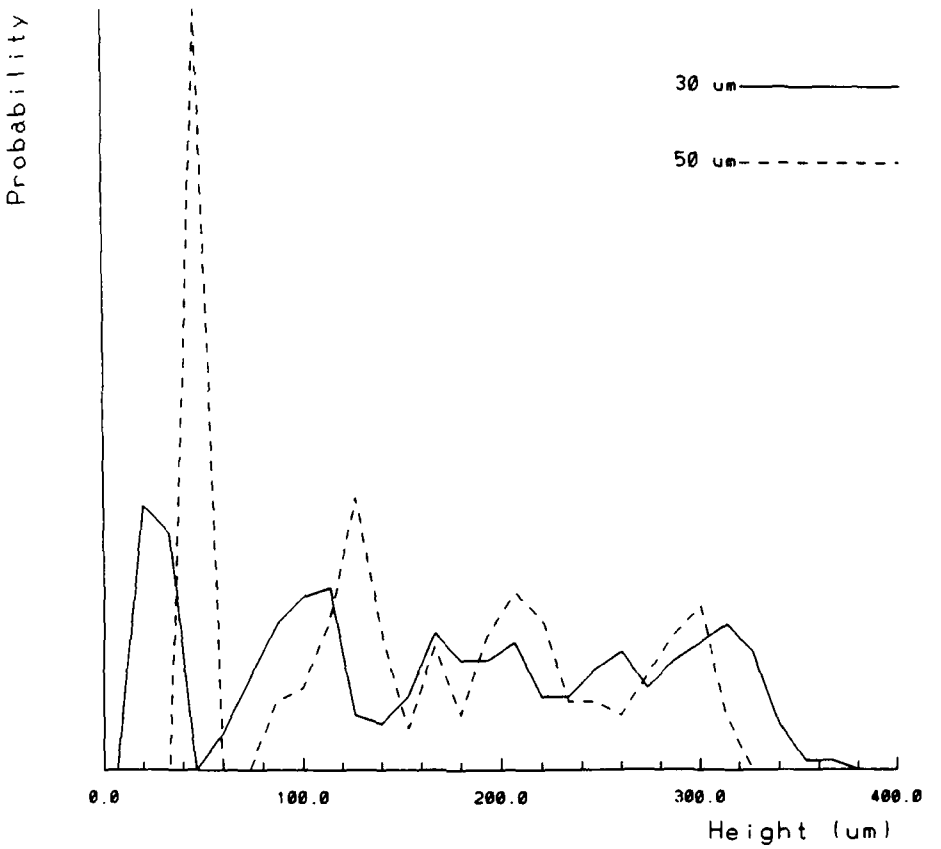

Figure 3 *Probability of finding a particle as a function of particle height above the floor of the container. 512 particles in total, 307 with radius of 30 μm (solid line), 205 with radius 50 μm (dashed line).*

SECURITY CLASSIFICATION OF THIS PAGE **UNCLASSIFIED**

DOCUMENT CONTROL DATA SHEET

REPORT NO.	AR NO.	REPORT SECURITY CLASSIFICATION
MRL-TR-89-30	AR-005-727	Unclassified

TITLE

Molecular dynamics simulation of sedimentation using the monotonic Lagrangian grid

AUTHOR(S)

R.A.J. Borg, D.A. Jones and S.G. Lambrakos

CORPORATE AUTHOR

DSTO Materials Research Laboratory
PO Box 50
Ascot Vale Victoria 3032

REPORT DATE	TASK NO.	SPONSOR
July 1989	DST 88/112	DSTO

FILE NO.	REFERENCES	PAGES
G6/4/8-3727	12	19

CLASSIFICATION/LIMITATION REVIEW DATE

CLASSIFICATION/RELEASE AUTHORITY
Chief, Explosives Division MRL

SECONDARY DISTRIBUTION

Approved for public release

ANNOUNCEMENT

Announcement of this report is unlimited

KEYWORDS

>Molecular dynamics Sedimentation

SUBJECT GROUPS 0046S 0072B

ABSTRACT

We describe the implementation of a new algorithm for tracking nearest-neighbours, the Monotonic Lagrangian Grid (MLG), in a recently developed Molecular Dynamics (MD) code used to study sedimentation phenomena. The MLG is a highly efficient algorithm for tracking particle positions which is based on the assignment of a set of indices to each particle in the system. These indices are ordered according to the relative positions of each of the particles, and are continuously updated as particles move around in space. Application of the MLG to the existing sedimentation code reduces the dependence of the CPU time on total particle number N from N^2 to N. This is the first application of the MLG to this type of system. Areas of the code are identified in which implementation of more advanced aspects of the MLG algorithm would allow efficient vectorization of the code and result in additional reduction in CPU time. Other applications for MD codes at MRL based on the MLG are also described.